TRANSDUCERS BASED ON NETWORKS OF EVOLUTIONARY PROCESSORS

Sandra Gómez Canaval

Victor Mitrana

and

Santiago Alonso Villaverde

ABSTRACT

We consider a new type of transducer that does not scan sequentially the input word. Instead, it consists of a directed graph whose nodes are processors which work in parallel and are specialized in just one type of a very simple evolutionary operation: inserting, deleting or substituting a symbol by another one. The computation on an input word starts with this word placed in a designated node, the input node, of the network and alternates between evolutionary and communication steps. The computation halts as soon as another designated node, the output node, is nonempty. The translation of the input word is the set of words existing in the output node when the computation halts. We prove that these transducers can simulate the work of generalized sequential machines on every input. Furthermore, all words obtained by a given generalized sequential machine by the shortest computations on a given word can also be computed by the new transducers. Unlike the case of generalized sequential machines, every recursively enumerable language can be the transduction defined by the new transducer of a very simple regular language. The same idea may be used for proving that these transducers can simulate the shortest computations of an arbitrary Turing machine, used as a transducer, on every input word. Finally, we consider a restricted variant of NEP transducer, namely pure NEP transducers and prove that there are still regular languages whose pure NEP transductions are not semilinear.

Keywords: Evolutionary rule; network of evolutionary processors; generalized sequential machine; NEP transducer.

Work supported by the Project TIN2011-28260-C03-03.

1. Introduction

Networks of evolutionary processors (NEP) form a class of highly parallel and distributed computing models inspired by and abstracted from the biological evolution. Moreover, networks of evolutionary processors have their origin in networks of language processors introduced in [4]. On the other hand, NEPs resemble a pretty common architecture for parallel and distributed symbolic processing, related to the Connection Machine [9] as well as the Logic Flow paradigm [7]. Also it is closely related to the tissue-like P systems [14] in the membrane computing area [17]. A rather informal idea of what a network of evolutionary processor is consists of a (complete) graph in which each node hosts a very simple processor called an evolutionary processor. By an evolutionary processor we mean a processor, which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). By an informal parallelism with the natural process of evolution, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node processor, which is specialized just for one of these evolutionary operations, acts on the local data and then local data becomes a mobile agent which can navigate in the network following a given protocol. Only that data which is able to pass a filtering process can be communicated. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies.

NEPs as language generating devices and problem solvers have been considered in [3] and [15], respectively. They have been further investigated in a series of subsequent works. NEPs as accepting devices and problem solvers have been considered in [13]; later on, a characterization of the complexity classes **NP**, **P**, and **PSPACE** based on accepting NEPs has been reported in [11]. Universal NEPs and some descriptional complexity problems are discussed in [10]. The reader interested in a survey of the main results regarding NEPs is referred to [12].

When a NEP is intended to be used as a universal problem solver, an important aspect is the phase of encoding the instance of the problem and that of decoding the solution. It is natural to ask whether or not this phase can be accomplished by a mechanism based on NEPs as well. Thus, it appears natural to consider a transducer based on the NEP structure. This is the aim of this note. We consider a new type of transducer that is not based on the sequential reading of the input word, but on NEPs. It consists of a directed graph whose nodes are evolutionary processors without filters. The computation on an input word starts with this word placed in a designated node, the input node, of the network and alternates between evolutionary and communication steps. The computation halts as soon as another designated node, the output node, is nonempty. The translation of the input word is the set of words existing in the output node when the computation halts. It is worth mentioning that several software implementations of NEPs have been reported, see, e.g., [5, 6, 16], most of them in JAVA. They encountered difficulties especially in the

implementation of filters. The communication of the transducer considered in this note is not regulated by filters, but the underlying graph is directed.

This work is organized as follows. First, we define the evolutionary operations and the ways in which they can be applied. It follows the definition of NEP transducers accompanied by an example for a better understanding. We prove that NEP transducers can simulate the work of generalized sequential machines (GSM) on every input word. Furthermore, for every word w given as input, NEP transducers are able to compute the set of all words obtained by a given GSM by the shortest computations on the same input word w. Unlike the case of GSM mappings, every recursively enumerable language can be the NEP transduction of the universal language over the one letter alphabet. We then discuss how the idea used in the proof of the last result might be extended to show that NEP transducers can simulate not only the work of GSM on every input word, but also that of any Turing machine as transducer. Finally, we consider a restricted variant of NEP transducer, namely pure NEP transducers having the working alphabet equal to the union of the input and output alphabets. We prove that there are still regular languages whose pure NEP transductions are not semilinear.

2. Basic definitions

We assume the reader to be familiar with fundamental concepts from Formal Language Theory, Chomsky hierarchy, Lindenmayer systems (L-systems), generalized sequential machine, regulated rewriting, which can be found in many textbooks, e.g., the handbook [18]. We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. Any finite sequence of symbols from an alphabet V is called a word over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. The length of a word x is denoted by $|x|$ while $|x|_a$ denotes the number of occurrences of the symbol a in x. For simplicity, we identify a regular language by its regular expression.

In the course of its evolution, the genome of an organism mutates by different processes. At the level of individual genes, the evolution proceeds by local operations (point mutations) which substitute, insert and delete nucleotides of the DNA sequence. In what follows, we define some rewriting operations that will be referred to as *evolutionary operations* since they may be viewed as linguistic formulations of local gene mutations. We say that a rule $a \to b$ with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

• If $\sigma = a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, a \text{ occurs in } w \\ \emptyset, \text{ otherwise.} \end{cases}$

Note that a rule as above is applied to all occurrences of the letter a in different copies of the word w. An implicit assumption is that arbitrarily many copies of w are

available.

• If $\sigma = a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, a \text{ occurs in } w \\ \emptyset, \text{ otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, a \text{ occurs in } w \\ \emptyset, \text{ otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : \ w = av\}, a \text{ occurs in } w \\ \emptyset, \text{ otherwise} \end{cases}$$

• If $\sigma = \varepsilon \to a \in Ins_V$, then $\sigma^*(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}$,
$\sigma^r(w) = \{wa\}$, $\sigma^l(w) = \{aw\}$.

We use $\alpha \in \{*, l, r\}$ for denoting the way of applying a deletion or insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. The note for the substitution operation mentioned above remains valid for insertion and deletion at any position. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by $M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

3. NEP transducers

A *transducer based on networks of evolutionary processors* (NEP transducer for short) is a 8-tuple $\gamma = (V, U, W, D, R, \alpha, x_I, x_O)$, where:

- V and U is the input and the output alphabet, respectively.
- W is the network alphabet, $(V \cup U) \subseteq W$.
- $D = (X_D, E_D)$ is a directed graph with the set of vertices X_D and the set of edges $E_D \subseteq (X_D \times X_D)$. D is called the *underlying graph* of the network.
- $R : (X_D \setminus \{x_O\}) \longrightarrow 2^{Sub_W} \cup 2^{Del_W} \cup 2^{Ins_W}$ is a mapping which associates with each node different than x_0 *the set of evolutionary rules* that can be applied in that node. Note that each node is associated only with one type of evolutionary rules, namely for every $x \in X_D$ either $R(x) \subseteq Sub_W$ or $R(x) \subseteq Del_W$ or $R(x) \subseteq Ins_W$ holds.
- The action mode of the rules of node x on the words existing in that node is given by $\alpha(x)$, where $\alpha : X_D \longrightarrow \{*, l, r\}$.
- $x_I, x_O \in X_D$ are the *input* and the *output* node of γ, respectively.

A *configuration* of a NEP transducer γ as above is a mapping $C : X_D \longrightarrow 2^{W^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node at a given moment.

When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules $R(x)$ associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

$$C'(x) = R(x)^{\alpha(x)}(C(x)) \text{ for all } x \in X_D.$$

When changing by a communication step, each node processor $x \in X_D$ sends one copy of each word it has to all the node processors y such that $(x, y) \in E_D$, and receives all the words sent by any node processor z such that $(z, x) \in E_D$. Clearly, the amount of words sent by each processor in a communication step could be huge which seems to be impractical from an "in silico" implementation point of view. However, there are well developed techniques in biochemistry which permit an exponential amplification of the genetic material in a linear number of steps. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = \bigcup_{(y,x)\in E_D} C(y) \text{ for all } x \in X_D.$$

Let γ be a NEP transducer, the computation of γ on the input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of γ defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_D$, $x \neq x_I$, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Note that the configurations are changed by alternative evolutionary and communication steps.

We say that a computation as above *halts* if there exists a configuration in which the output node x_O contains at least a word over W. Given a NEP transducer γ as above and an input word $w \in V^*$, we say that γ translates w into $z \in U^*$ if the computation of γ on w halts with z in the output node. Formally, we define the transduction function of γ denoted by Θ_γ as follows:

$$\Theta_\gamma(w) = C^k(x_O) \cap U^*,$$

provided that the computation of γ on w halts after $k \geq 1$ steps. In other words, $\Theta_\gamma(w)$ collects all possible words $z \in U^*$ such that w is translated by γ into z. If the computation of γ on w never halts, then $\Theta_\gamma(w) = \emptyset$. Furthermore, if L is a language over V, we set $\Theta_\gamma(L) = \bigcup_{w\in L} \Theta_\gamma(w)$.

In the sequel, we present a few results on the computational power of NEP transducers starting with an example in order to make clearer the concepts defined above.

Example 1 *Let γ be a NEP transducer having the unary alphabet $\{a\}$ as the input alphabet, and $\{a, b, c\}$ as the common working and output alphabet. We prefer to give the description of γ by means of the graph in Figure 1 in which the action mode of each rule is specified between parentheses.*

We claim that $\Theta_\gamma(a^n) \cap a^+b^+c^+ = a^nb^nc^n$ for every $n \geq 1$. Indeed, on one hand, obviously $a^nb^nc^n$ belongs to $\Theta_\gamma(a^n)$. This can be accomplish as follows: after inserting c to the right of the input word a^n, the word enters successively the nodes x_1, x_2, x_3, and x_4 such that at the end of this process a word $a^{n-1}cabc$ goes out from x_4 and enters x_1. This process can be resumed such that after n such cycles the word $ca^nb^nc^n$ goes out from x_4, enters x_5, where the leftmost c is deleted and eventually the word $a^nb^nc^n$ enters x_O.

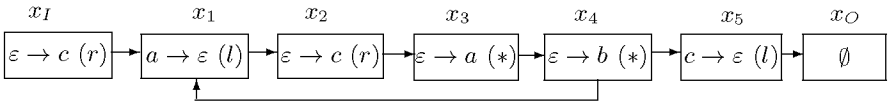

Figure 1.

On the other hand, the computation of γ cannot halt on the input word a^n before c becomes the leftmost symbol of the current word. Please note that each insertion of c is always done in the right-hand end of the current word. If b is inserted to the left of the leftmost c at some step, then that word can never reach the output node. This means that at least n occurrences of a have been deleted from the left-hand end of the input word. Furthermore, after n occurrences of a have been iteratively deleted in x_1, the computation halts with the output node containing a set of words with exactly n occurrences of a, b, and c, respectively. This set includes $a^n b^n c^n$. It is worth mentioning that, if at a given step during the computation an a is inserted to the left of the leftmost c (in the node x_3), the obtained word will never reach the output node as the computation halts before it might have reached the output node.

In conclusion, $\Theta_\gamma(\{a^n \mid n \geq 1\}) \cap a^+b^+c^+ = \{a^n b^n c^n \mid n \geq 1\}$, holds. □

As a NEP transducer is a language translator, it is natural to compare it with other language translators, especially with the most widely known transducer, namely the finite-state transducer. Actually, we consider here a subclass of finite-state transducers, that is the *generalized sequential machines* (*GSM* for short). *GSM* are applied in many areas of computer science, e.g., in text processing (pattern matching, indexing, compression), natural language processing (recognition, synthesis), image processing (filtering, compression). A *GSM* is a 6-tuple $M = (Q, V, U, T, q_0, F)$, where Q is a finite set of states, V and U are the input and output alphabets, respectively, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and T is a finite set of transitions of the form $qa \to xs$, where $a \in V$, $q, s \in Q$, $x \in U^*$. A configuration of M is a word of the form yqx with $y \in U^*$ (the current contents of the output tape of M), $q \in Q$ (the current state of M), and $x \in V^*$ (the remaining part of the input word). We write $yqax \longrightarrow yzsx$ iff $qa \to zs \in T$. The reflexive and transitive closure of the relation $\longrightarrow$ is denoted by $\longrightarrow^*$. The transduction defined by M is usually called *GSM* mapping and is defined as follows:

$$T_M(x) = \{y \mid q_0 x \longrightarrow^* ys, s \in F\},\ x \in V^*, \qquad \text{and} \qquad T_M(L) = \bigcup_{x \in L} T_M(x),\ L \subseteq V^*.$$

We further define: $\mu T_M(x) = \{y \in T_M(x) \mid |y| \leq |z| \text{ for all } z \in T_M(x)\},\ x \in V^*,$

$$\mu T_M(L) = \bigcup_{x \in L} \mu T_M(x),\ L \subseteq V^*.$$

The next results establish a relationship between the computational power of the two types of transducers.

Theorem 3.1 *Let M be a GSM with the input alphabet V.*
1. There exists a NEP transducer γ_1 with the input alphabet V such that $\mu T_M(w) = \Theta_{\gamma_1}(w)$, for every $w \in V^*$.

2. There exists a NEP transducer γ_2 with the input alphabet V such that $T_M(w) = \Theta_{\gamma_2}(w)$, for every $w \in V^*$.

Proof. 1. Let $M = (Q, V, U, T, q_0, F)$ be a GSM and $L \subseteq V^*$. We define the NEP transducer $\gamma_1 = (V, U, W, D, R, \alpha, x_I, x_O)$ as follows:

$$
\begin{aligned}
W &= V \cup U \cup Q, \\
D &= (X_D, E_D), \text{ with} \\
X_D &= \{x_I, x_{po}, x_O\} \cup \{[qx] \mid q \in Q, x \in V^*\} \cup \\
&\qquad \{[asx] \mid qa \to xs \in T, \text{ for some } q \in Q\} \\
E_D &= \{(x_I, [q_0])\} \cup \{([q], [asx]) \mid qa \to xs \in T\} \cup \{([asx], [sx]) \mid a \in V\} \cup \\
&\qquad \{([say], [sy]) \mid a \in V, y \in V^*\} \cup \{([q], x_{po}) \mid q \in Q\} \cup \{(x_{po}, x_O)\}.
\end{aligned}
$$

The evolutionary rules in each processor and their action modes are given in Table 3.

Node	R	α
x_I	$\{\varepsilon \to q_0\}$	r
x_{po}	$\{q \to \varepsilon \mid q \in F\}$	l
$[q], q \in Q$	$\{s \to q \mid s \in Q\}$	$*$
$[asx], qa \to xs \in T$, for some $q \in Q$	$\{a \to \varepsilon\}$	l
$[qax], a \in V, q \in Q, x \in V^*$	$\{\varepsilon \to a\}$	r

Table 1: The evolutionary rules in the nodes of γ_1.

Assume that $q_0 w \longrightarrow^* yqaz \longrightarrow yy'sz$ is a computation in M. We now analyze a computation in γ on the input word w. Initially, the symbol q_0 is inserted in the right-hand end yielding wq_0 which enters $[q_0]$. Here the symbol q_0 is replaced by itself and a copy of wq_0 arrives in each node $[bpu]$ for all $q_0 b \to up \in T$. As wq_0 was present in the node $[q_0]$, we may inductively assume that $azqy$ is present in the node $[q]$. Now a copy of this word enters each node $[crv]$ for all $qc \to vr \in T$, hence also $[asy']$. Note that each copy that arrives in a node $[crv]$ with $a \neq c$ will disappear while the first a is deleted in the node $[asy']$. The new word zqy enters now $[sy']$. We assume that $y' = a_1 a_2 \dots a_j$ for some $j \geq 1$, $a_i \in W$, $1 \leq i \leq j$. The word zqy will enter successively in the nodes $[sa_1 a_2 \dots a_j]$, $[sa_2 \dots a_j]$, $\dots$, $[sa_j]$, where the symbols $a_1, a_2, \dots, a_j$ are inserted successively in the right-hand end of the current word. Finally, the word $zqyy'$ enters $[s]$, where q is substituted with s. Note that if $y' = \varepsilon$, the word zqy arrives directly in $[s]$. A copy of every word which is obtained in an evolutionary step in a node $[q]$, $q \in Q$, enters the node x_{po}. However, every such word which does not start with a symbol s, with $s \in F$, will disappear in x_{po}. On the other hand, if it starts with such a symbol, it enters x_O after loosing its first symbol.

By these explanations we may infer that $\Theta_{\gamma_1}(w) \subseteq T_M(w)$ for every $w \in V^*$. On the other hand, for every word $y \in T_M(w)$ the NEP transducer γ_1 would need at least $2|w| + |y| + 1$ evolutionary steps to compute it on the input word w. Therefore, only

the shortest words in $T_M(w)$ can be computed by γ_1 on w. This concludes the proof of the first statement.

2. The rough idea in the proof of the first statement is that the underlying graph of γ_1 has a uniquely identified subnetwork for simulating each transition in M. The NEP transducer γ_1 may be easily modified such that when the computation halts on an input word w, the output node contains exactly all words from $T_M(w)$. If we denote by $k = max\{|z| \mid qa \to zs \in T\}$, then γ_1 can be modified as follows. The subnetwork responsible for each transition $qa \to zs$ is either left unchanged, provided that $|z| = k$, or is modified in its part which inserts z in the right-hand end by adding a number of $k - |z|$ substitution nodes that have no effect, e.g., they substitute the symbol representing the current state by itself. It immediately follows that $T_M(w) = \Theta_{\gamma_2}(w)$, for every $w \in V^*$, where γ_2 is obtained from γ_1 by the aforementioned modifications. □

It is known that GSM mappings preserve regular languages. The situation is very different with NEP transductions as shown already in Example 1. Furthermore,

Theorem 3.2 *For every recursively enumerable language $L \subseteq T^*$, there exist a symbol $a \notin T$ and a NEP transducer γ such that $L = \Theta_\gamma(a^+)$ holds.*

Proof. The general idea of the proof is to simulate the leftmost derivation in a random context grammar. A *random context* grammar (see [19]) is a structure $G = (N, T, S, P)$, where N and T are the alphabets of nonterminals and terminals, respectively, and $S \in N$ is the grammar's axiom. Further on, P is a finite set of rules written as a triple $(A \to \beta, Q, F)$, where $A \to \beta$ is a classic Chomsky context-free rule while Q and F are subsets of N. Given a word $y \in (N \cup T)^+$ we say that y is rewritten by a rule $(A \to \beta, Q, F) \in P$ in the leftmost manner [2] if the following conditions are satisfied:

- $y = y_1 A y_2$, all symbols from Q appear in $y_1 y_2$ and no symbol from F appears in $y_1 y_2$.
- There is no other rule $(A' \to \beta', Q', F') \in P$ such that $y = y_1' A' y_2'$, $|y_1'| < |y_1$, all symbols from Q' appear in $y_1' y_2'$ and no symbol from F' appears in $y_1' y_2'$.

Obviously, the result of rewriting y by the rule $(A \to \beta, Q, F)$ in the leftmost manner as above is $z = y_1 \beta y_2$. Formally, we write $y \Longrightarrow_G z$. The language generated by G in the leftmost manner is $L(G) = \{w \in T^* \mid S \Longrightarrow_G^* w\}$. It is known (see [2]) that for every recursively enumerable language L there exists a random context grammar such that $L = L(G)$.

Let $m = \max\{|\beta| \mid (A \to \beta, Q, F) \in P, \text{ for some } A \in N, Q, F \subseteq N\}$. We consider the new random context grammar $G' = (N, T \cup \{c\}, S, P')$, where c is a new symbol not in $N \cup T$ and $P' = \{(A \to \beta c^{m-|\beta|}, Q, F) \mid (A \to \beta, Q, F) \in P\}$. If $L(G)$ and $L(G')$ is the language generated by G and G', respectively, in the leftmost manner, then $L(G) = pr_T(L(G'))$, where pr_T is a morphism that erases the letter c and leaves unchanged all letters from T. Note that all sentential forms of G' obtained after k derivation steps are of the same length. This fact will turn out to be useful in our further reasoning.

We now informally describe how a NEP transducer γ can translate any input a^n into all sentential forms of G' obtained after n derivation steps. From these words, γ will further squeezes out the words of $L(G)$ generated in exactly n steps. The work of γ is divided into the following phases:

Phase 1.
P1.1. In the node x_I a working symbol $\# \notin (N \cup T \cup \{c\})$ is appended to the input word, say a^n and the obtained word enters x_{start}.
P1.2. In x_{start}, the symbol S is inserted in the right-hand end. Now, the current word is $a^n \# S$ which enters the node y_ε.

Phase 2. This phase is accomplished by a subnetwork of γ defined as shown in Table 2 in which the last column contains the set of direct successors of that node. To this aim, let Σ be the set of all words u containing symbols of the form (A, i), $A \in N$, $i = 1, 2$ such that $|u|_{(A,1)} + |u|_{(A,2)} \leq 1$ for all $A \in N$. Note that $\varepsilon \in \Sigma$. Moreover, in Table 2, A ranges over N, b ranges over $T \cup \{c\}$ and u ranges over Σ.

Node	R	α	Successors
y_u, $u \in \Sigma$	$\{\# \to \#\}$	$*$	$\{z_u^b \mid b \in T \cup \{c\}\} \cup \{\langle check\_end, u\rangle\} \cup \{x_{u'}^A \mid u' = \begin{cases} (A,1)u, \text{ if } (A,i),\ i=1,2 \text{ does not appear in } u \\ (A,2)u_1u_2, \text{ if } u = u_1(A,i)u_2, \text{ for some } i = 1,2 \end{cases}\}$
x_u^A	$\{A \to \varepsilon\}$	r	$\{\bar{x}_u^A\}$
$\bar{x}_u^A$	$\{\varepsilon \to A\}$	l	$\{y_u\}$
z_u^b	$\{b \to \varepsilon\}$	r	$\{\bar{z}_u^b\}$
$\bar{z}_u^b$	$\{\varepsilon \to b\}$	l	$\{y_u\}$
$\langle check\_end, u\rangle$	$\{\# \to \varepsilon\}$	r	$\{\langle end, u\rangle\}$
$\langle end, u\rangle$	$\{\varepsilon \to \#\}$	l	$\{\langle simulate, u\rangle\}$

Table 2: The nodes of the subnetwork of γ that accomplishes Phase 2.

By this construction, we infer that if the node $\langle simulate, u\rangle$, for some $u = (A_1, j_1)(A_2, j_2)\dots(A_k, j_k)$, is nonempty at the end of Phase 2, then the suffix after the symbol $\#$ of the ancestor of every word in $\langle simulate, u\rangle$ that was in the beginning of Phase 2 in y_ε satisfied the following conditions:

- For any $1 \leq i \leq k-1$ there is an occurrence of the nonterminal A_i before the first occurrence of A_{i+1}.
- For any $1 \leq i \leq k$ the number of occurrences of A_i is i, provided that $i = 1$, or at least 2, providing that $i = 2$.

Let w be such a suffix as above. This means that a rule $(A \to \beta, Q, F) \in P'$ is applicable to w if the following statements hold:

– There exists an i such that the i-th symbol of u is (A, j) for some $j \in \{1, 2\}$.

– For every nonterminal $B \in Q \setminus \{A\}$, there exists $1 \leq p \leq 2$ such that (B, p) does appear in u. If $A \in Q$, then j from above must be 2.
– For every nonterminal $B \in F \setminus \{A\}$, (B, p) does not appear in u for any p. If $A \in F$, then j from above must be 1.
– The number i from above is the minimal.

Phase 3. Note that every suffix w as above is now a prefix of the words in $\langle simulate, u\rangle$. A subnetwork of γ that starts with $\langle simulate, u\rangle$ actually applies all applicable rules to every w as above to different copies of the words in $\langle simulate, u\rangle$. This subnetwork will eventually translate every current word $wa^k\#$ from $\langle simulate, u\rangle$ into a word $a^k\#w'$ such that w' is obtained in G' by applying a rule to w in the leftmost manner. Let us assume that the applied rule is $(A \to \beta, Q, F)$. The subnetwork rotates from left to right all symbols of the current word until the first occurrence of A is met. This is deleted and then β is inserted letter by letter in the right-hand end. Afterwards, the rotation continues until an a is met. This a is deleted and all the words are now sent to y_ε and the computation resumes with Phase 2. Note that all the words sent to y_ε at the end of Phase 3 are actually of the form $a^k\#w$ where w is a sentential form of G' obtained after $n-k$ derivation steps.

Phase 4. This phase starts when a word in y_ε starts with $\#$. In this case, another subnetwork of γ rotates from right to left the current word and deletes all occurrences of the symbol c. If the current word does not contain any nonterminal, then γ outputs the word after deleting the rightmost symbol $\#$. If a nonterminal is met during this process, that word gets stuck in some node.

Consequently, $L(G) = \Theta_\gamma(a^+)$, and we are done. □

The idea of this proof may be very informally expressed as follows: the input word (actually its length) indicates the number n of leftmost derivations in the grammar G' simulated by the NEP transducer γ. It is known, see, e.g., [20], that a GSM mapping can simulate a single derivation step of any Chomsky grammar. The proof of the second statement of Theorem 3.1 might suggest that one can try to consider a NEP transducer that simulates the iteration of a GSM which in its turn simulates the work of an arbitrary grammar. It is worth mentioning that this does not immediately follow from the facts aforementioned as in the process of simulation of a derivation step in an arbitrary grammar a NEP transducer needs to rotate the sentential form, hence the shortest ones will be completely rotated before the other ones. This can be avoided as in the previous proof by using a new symbol c but in the case of an arbitrary grammar, arbitrary many occurrences of this symbol may split the right-hand side of an applicable rule which is now impossible to be applied.

A closer look to the previous proof suggests an indication how the idea of this proof may be used for proving that NEP transducers can simulate the shortest computations of a Turing machine as transducer on every input. More precisely, we assume without loss of generality that any Turing machine we consider here has a semi-infinite tape (bounded to the left) and makes no stationary moves; the computation of such a machine is described in [18]. Formally, a nondeterministic Turing machine is a

construct $M = (Q, V, U, \delta, q_0, B, F)$, where Q is a finite set of states, V is the input alphabet, U is the tape alphabet, $V \subset U$, q_0 is the initial state, $B \in U \setminus V$ is the "blank" symbol, $F \subseteq Q$ is the set of final states, and δ is the transition mapping, $\delta : (Q \setminus F) \times U \to 2^{Q \times (U \setminus \{B\}) \times \{R,L\}}$. As a transducer, the Turing machine M defines the function $f_M : V^* \longrightarrow (U \setminus \{B\})^*$, where $f_M(w)$ is the set of all words on the tape at the end of any computation of M on w that enters a final state. Formally, $f_M(w) = \{z \mid q_0 w \models^* qz, q \in F\}$. The set of all words defined by M by the shortest computations on w is

$$\mu f_M(w) = \{z \mid q_0 w \models^n qz \text{ and for every computation } q_0 w \models^m q'z', q' \in F, \\ m \geq n \text{ holds.}\}$$

The idea of the last proof may be easily adapted (this is left to the reader) to prove the next result.

Theorem 3.3 *Let M be a Turing machine as transducer with the input alphabet V. There exists a NEP transducer γ with the input alphabet V such that $\mu f_M(w) = \Theta_\gamma(w)$, for every $w \in V^*$.*

One of the key points of all these proofs is to use at least one symbol which belongs neither to the input nor the output alphabet. It may act either as a marker for the beginning/end of the input word or as a "dummy" symbol necessary for keeping the same length.

A natural problem is to consider a restricted variant of NEP transducer, namely NEP transducers having the working alphabet equal to the union of the input and output alphabets. We say that a NEP transducer satisfying this condition is *pure*. The statements from Theorem 3.1 remain valid as long as the input alphabet and the output alphabet of the *gsm* are disjoint. However, there are still regular languages whose pure NEP transductions are neither context-free (see 1) nor semilinear (see the next theorem).

Theorem 3.4
1. For any D0L language L there exist a symbol a and a pure NEP transducer γ such that $L = \Theta_\gamma(a^+)$.
2. For any regular language L there exist a symbol a and a pure NEP transducer γ such that $L = \Theta_\gamma(a^+)$.

Proof. 1. The construction of γ is rather simple and may be informally described as follows. The axiom w of the D0L system $G = (V, w, h)$, $a \notin V$, is appended to the input word. Then an a is deleted and the application of the morphism h is simulated by rotating from right to left all symbols from V until an a is met. Now, this a is deleted and a new application of h is simulated by rotating from left to right all symbols from V. The process resumes until all occurrences of a have been deleted.

2. We make use of the following characterization of the class of regular languages [8]. A *prefix grammar* defines a language in a pure way: all sentential forms belong to the language. The derivation process starts from a specified finite sets of words

and rewrites prefixes of the words already generated. We briefly recall from [8] the definition of a prefix grammar. A prefix grammar is a triple $G = (V, S, P)$, where V is an alphabet, S is a finite set of words over V, and P is a finite set of productions of the form $u \to v$ with $u, v \in V^+$. The production $u \to v$ may be applied to a word x yielding y, if $x = ux'$ and $y = vx'$. The language generated by G is the set of all words that can be obtained from the words of S by applying arbitrarily many times (including 0) the rules from P. In [8] one proves that a language is regular if and only if it is generated by a prefix grammar.

We now prove that any language generated by a prefix grammar $G = (V, S, P)$ is the NEP transduction of the language defined by the regular expression a^+. Without loss of generality, we may assume that S contains exactly one word, say w. As the proof is pretty simple, we prefer to present it intuitively; the reader interested in a formal proof may get it just following our informal explanations. Clearly, the output as well as the working alphabet of the NEP transducer γ is V which does not contain the symbol a. On every input a^n, γ works as follows:

- Inserts the word w in the beginning of the input word.
- Deletes an a from the right-hand end of the current word.
- Simulates the application of a rule $u \to v$ from P by deleting successively all symbols of the prefix u and inserts successively all symbols in the reverse order in which they appear in v in the left-hand end of the current word. If it is not possible to delete all letters of u, the computation gets stuck. As in the proof of the second statement of Theorem 3.1, "dummy" nodes are added such that the simulation of each rule takes the same number of evolutionary steps.
- Resumes the process described above by deleting another symbol a until all symbols a have been deleted.

By these explanations, it easily follows that all words generated by G in n steps will be in the output node of γ when it halts on the input word a^n. □

References

[1] P.P. Alarcón, F. Arroyo, V. Mitrana, Networks of polarized evolutionary processors as problem solvers. In: *16th Annual Conference on Advances in Knowledge-Based and Intelligent Information and Engineering Systems - KES 2012*, IOS Press 2012, 807–815.

[2] A.B. Cremers, H.A. Maurer, O. Mayer, A note on leftmost restricted random context grammars, *Inform. Proc. Letters* **2** (1973), 31–33.

[3] J. Castellanos, C. Martín-Vide, V. Mitrana, J.M. Sempere, Networks of evolutionary processors, *Acta Inform.* **39** (2003), 517–529.

[4] E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors. In: *New Trends in Formal Languages*, LNCS 1218, Springer, 1997, 299–318.

[5] M.A. Diaz, L.F. de Mingo, N. Gómez Blas, J. Castellanos, Implementation of massive parallel networks of evolutionary processors (MPNEP): 3-colorability problem. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, Studies in Computational Intelligence 129, Springer, 2008, 399–408.

[6] M.A. Diaz, L.F. de Mingo, N. Gómez Blas, Networks of evolutionary processors: Java Implementation of a threaded processor, *International Journal Information Theories & Applications* **15** (2008), 37–43.

[7] L. Errico, C. Jesshope, Towards a new architecture for symbolic processing. In: *Artificial Intelligence and Information-Control Systems of Robots '94*, 1994, 31–40.

[8] M. Frazier, C. David Page Jr., Prefix grammars: An alternative characterization of the regular languages, *Inform. Proc. Letters* **5** (1994), 67–71.

[9] W.D. Hillis, *The Connection Machine*. MIT Press, Cambridge, 1979.

[10] F. Manea, C. Martín-Vide, V. Mitrana, On the size complexity of universal accepting hybrid networks of evolutionary processors, *Mathematical Structures in Computer Science* **17** (2007), 753–771.

[11] F. Manea, M. Margenstern, V. Mitrana, M.J. Pérez-Jiménez, A new characterization of NP, P, and PSPACE with accepting hybrid networks of evolutionary processors, *Theory Comput. Syst.* **46** (2010), 174–192.

[12] F. Manea, C. Martín-Vide, V. Mitrana, Accepting networks of evolutionary word and picture processors: A survey. In: *Scientific Applications of Language Methods*, Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory, World Scientific, 2010, 523–560.

[13] M. Margenstern, V. Mitrana, M.J. Pérez-Jiménez, Accepting hybrid networks of evolutionary processors. In: *10th International Workshop on DNA Computing, DNA 10*, Lecture Notes in Computer Science 3384, Springer-Verlag, Berlin, 2004, 235–246.

[14] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón, A new class of symbolic abstract neural nets: tissue P systems. In: *8th Annual International Conference, Cocoon 2002*, Lecture Notes in Computer Science, 2387, 2002, 290–299.

[15] C. Martín-Vide, V. Mitrana, M.J. Pérez-Jiménez, F. Sancho-Caparrini, Hybrid networks of evolutionary processors. In: *Genetic and Evolutionary Computation - GECCO 2003*, Lecture Notes in Computer Science 2723 Springer-Verlag, Berlin 2003, 401–412.

[16] C.B. Navarrete, M. Echeanda, E. Anguiano, A. Ortega, J.M. Rojas, Parallel simulation of NEPs on clusters. In: *Proc. IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - WI-IAT 2011*, IEEE Computer Society 2011, 171–174.

[17] Gh. Păun, *Membrane computing. An introduction.* Springer-Verlag, Berlin, 2002.

[18] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, Springer-Verlag, Berlin, vol. I–III, 1997.

[19] A.P.J.van der Walt, Random context grammars. In: *Proc. IFIP Congress*, North Holland Publ. Co., 1970, 66–68.

[20] D. Wood, Iterated a-NGSM maps and Γ-systems, *Inform. Control* **32** (1976), 1–26.